



The NXOpen API : How it Works

Alasdair Mackintosh
UGS Architecture



- ▶ Provides a modern, Object Oriented interface to NX
- ▶ Provides the same interface in:
 - ▶ Java
 - ▶ Microsoft .NET
 - ▶ C++
- ▶ Supports remote (client/server) access
- ▶ Provides documentation in three formats
- ▶ Supports existing User Function interface in .NET and Java



Anatomy of a Simple Class : Line



- ▶ A Line has a start point and an end point.

- ▶ **Java**

```
public interface Line extends Curve {  
    /** Returns the start point of the line  
    <br> License requirements: None.<br> */  
  
    public Point3d startPoint() throws NXException, RemoteException;
```

- ▶ **.NET**

```
public class Line: Curve {  
    /// <summary> Returns the start point of the line </summary>  
    /// <remarks> License requirements: None. </remarks>  
  
    public unsafe Point3d StartPoint { ...
```

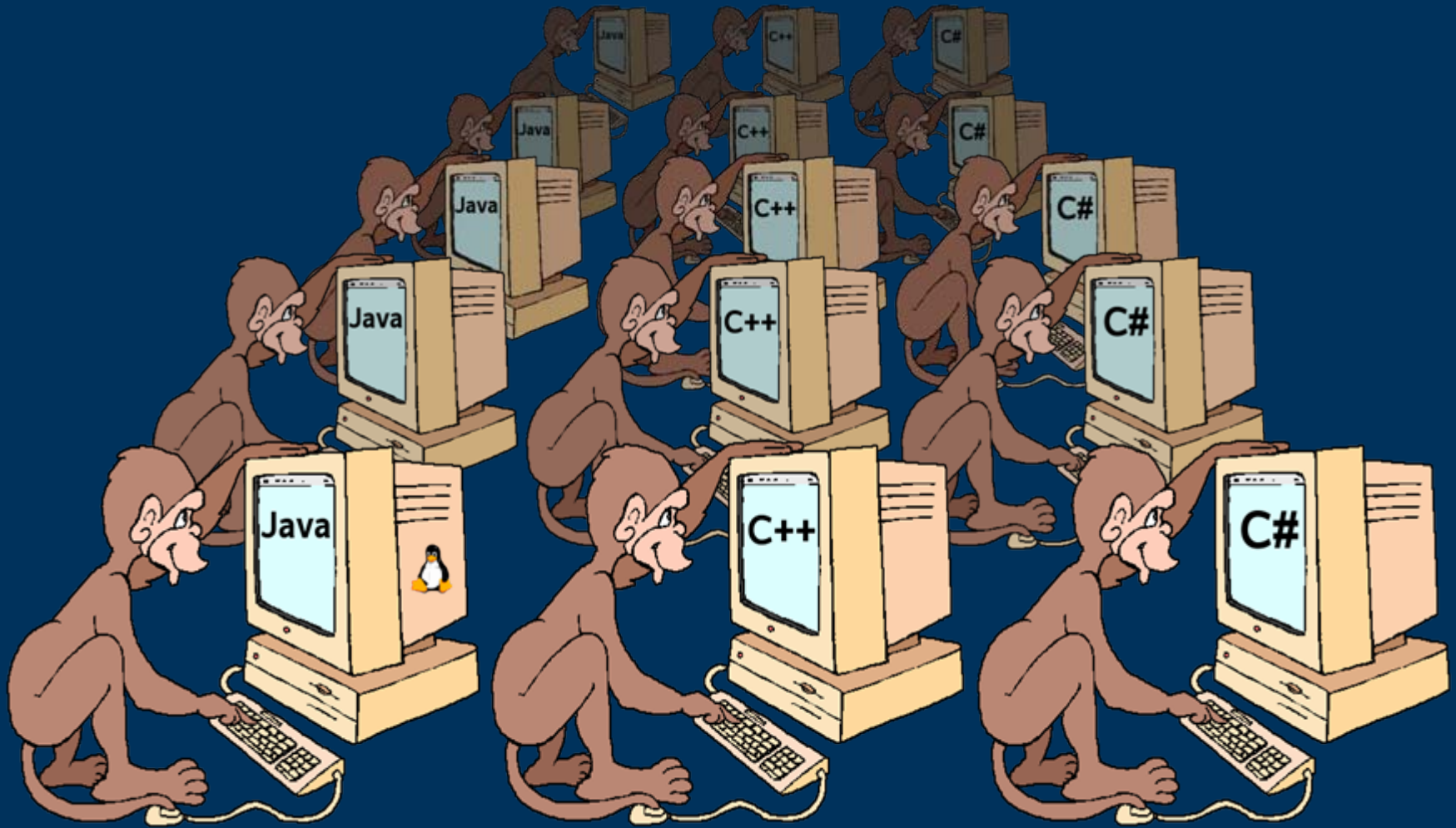
- ▶ **C++**

```
class NXOPENCPPEXPORT Line : public Curve {  
    /**Returns the start point of the line  
    <br> License requirements : None */  
  
    public: NXOpen::Point3d StartPoint();
```

- ▶ How do we support 700+ complex classes in three languages?



How do we do it?





- ▶ The interface for each class is defined in a an interface definition file. Internally called a “JA” file.
- ▶ Line.ja
- ▶

```
class Line : Curve
{
    /** the start point of the line */
    [no_license]
    [version_created("3")]
    extern API_PROPERTY int JA_LINE_get_start_point
    (
        tag_t line API_THIS,
        PNT3_p_t start_point API_RESULT /** */
    );
}
```



Interface and Implementation



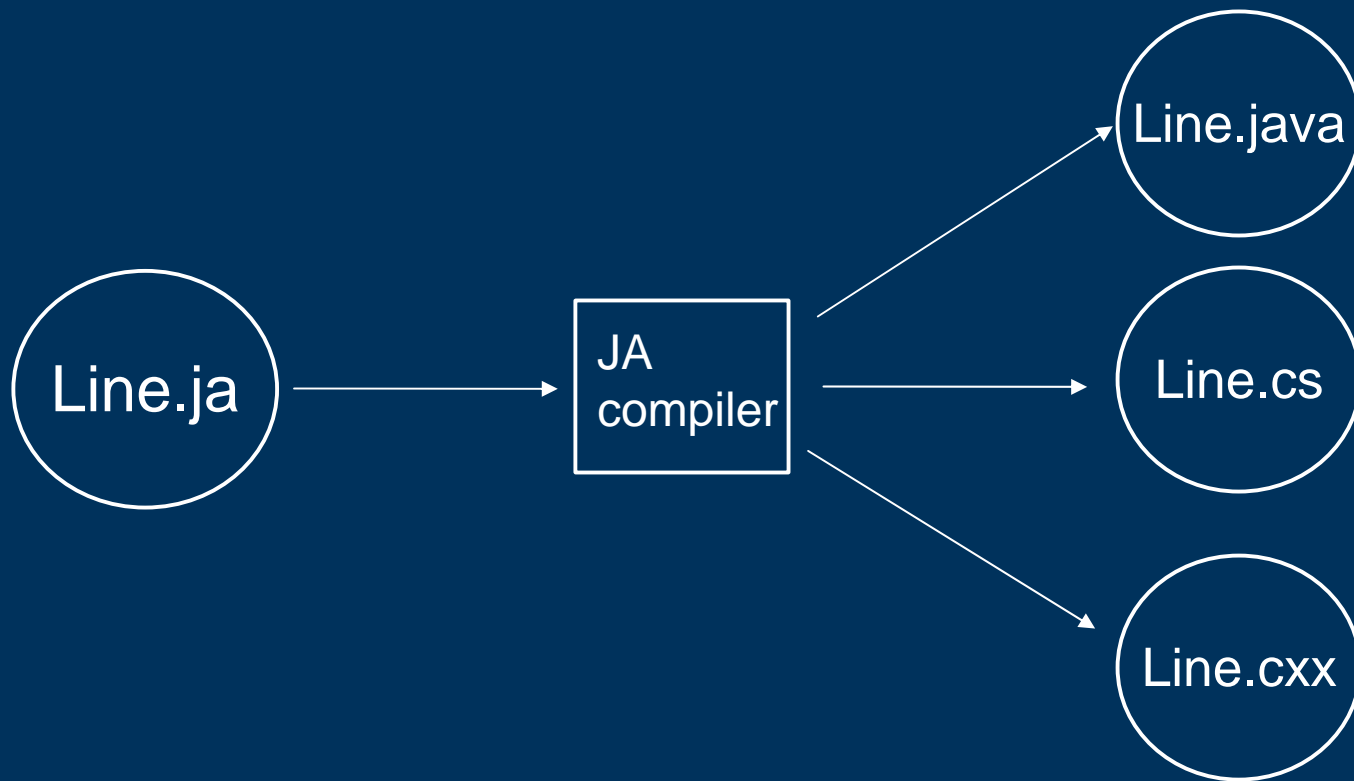
- ▶ The developer writes an interface definition in the JA file, and then writes an implementation of that interface in an internal NX C/C++ source file.

```
▶ extern int JAX_LINE_get_start_point  
(  
    tag_t line,  
    PNT3_p_t start_point  
)  
{  
    double line_def[3];  
    ES_GetLineDef(line, line_def);  
    PNT3_GetPoint(line_def, start_point);  
    return 0;  
}
```

Top Secret

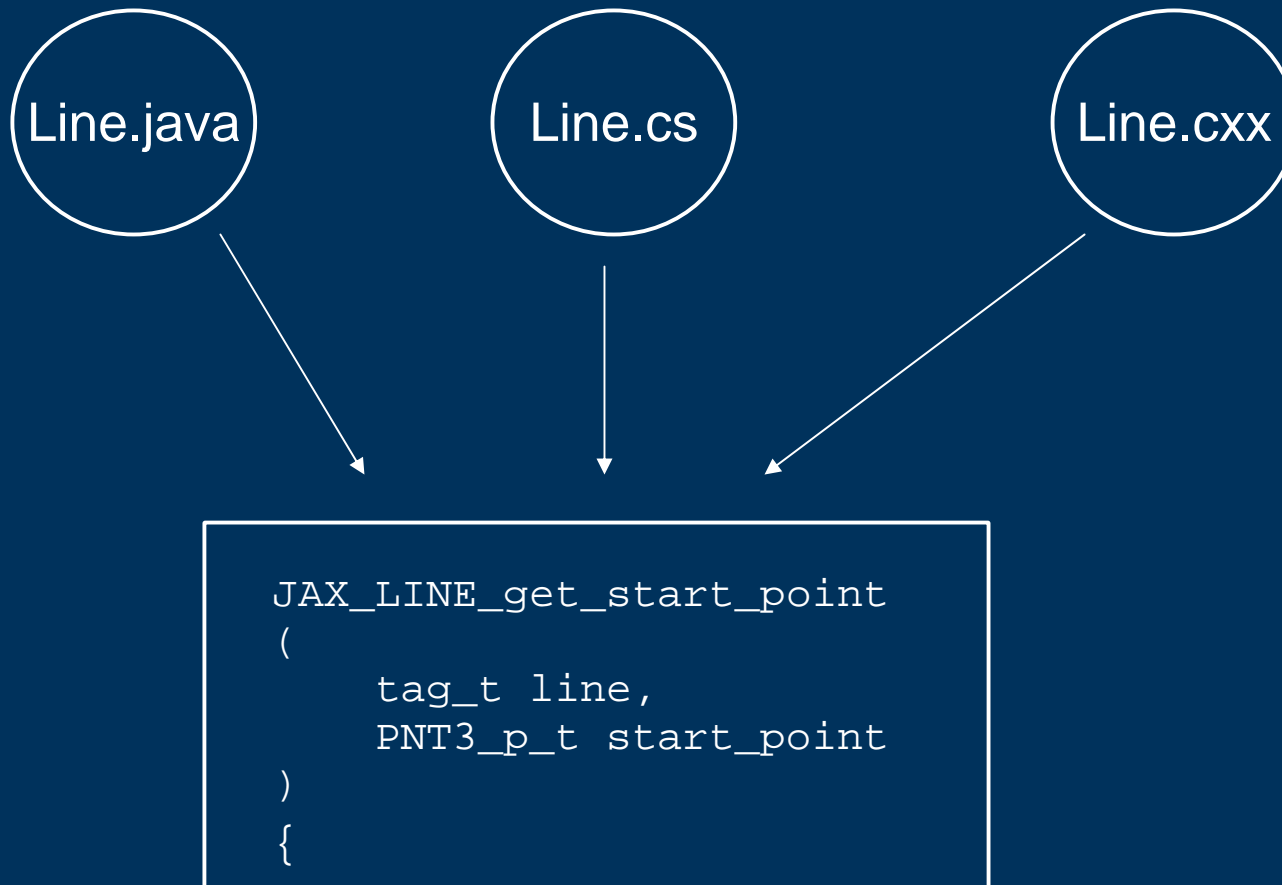


- The JA file is then compiled to produce the Java, .NET and C++ wrappers.





- Each Language invokes the same underlying API





- ▶ The Compiler handles all of the supported data types:
 - ▶ Fundamental types
 - ▶ Integers, doubles, boolean (true/false)
 - ▶ Strings (Unicode and locale)
 - ▶ Enumerated types (Create, Unite, Intersect)
 - ▶ API Types
 - ▶ Point, Vector, Matrix
 - ▶ Structures
- ▶ Each type is handled appropriately for the target language.



- ▶ Produces an API matched to the target language.
For example:
 - ▶ Uses `std::vector` in C++, arrays in Java and .NET
 - ▶ Uses enums in C++ and .NET, “static final int” in Java
 - ▶ Special handling for output arguments in Java
 - ▶ Special class for Unicode strings in C++
 - ▶ Uses properties in .NET, methods in C++ and Java
- ▶ The developer writing the JA file does not need to know the details of each language.



- ▶ Compiler handles all details of interface between API language and NX core.
 - ▶ Each data type is correctly handled.
 - ▶ Translates from managed to native code.
 - ▶ Produces correct JNI code for Java API
 - ▶ Uses .NET mechanism for invoking native functions
- ▶ Handles error codes and exceptions in the NX core, and throws NXException objects to callers of the NXOpen API.



- ▶ Remote Procedure Calls are supported in .NET and Java
- ▶ Uses standard protocols (RMI, .NET Remoting Mechanism)
- ▶ JA Classes are designed to support remoting
 - ▶ No constructors
 - ▶ All factory classes accessible from Session or Part class
- ▶ Compiler handles details
 - ▶ Enforces above rules
 - ▶ Correct inheritance to support remoting.
 - ▶ All structures serializable



- ▶ Developers provide documentation in the JA file

- ▶ **Line.java**

- ▶

```
class Line : Curve
{
    /** the start point of the line */
    extern API_PROPERTY int JA_LINE_get_start_point
```

- ▶ Compiler produces documentation suitable for each language

 StartPoint	Returns the start point of the line
--	-------------------------------------

- ▶ Classes and methods named correctly for each language
- ▶ Hyperlinks between classes.



User Function in Java and .NET



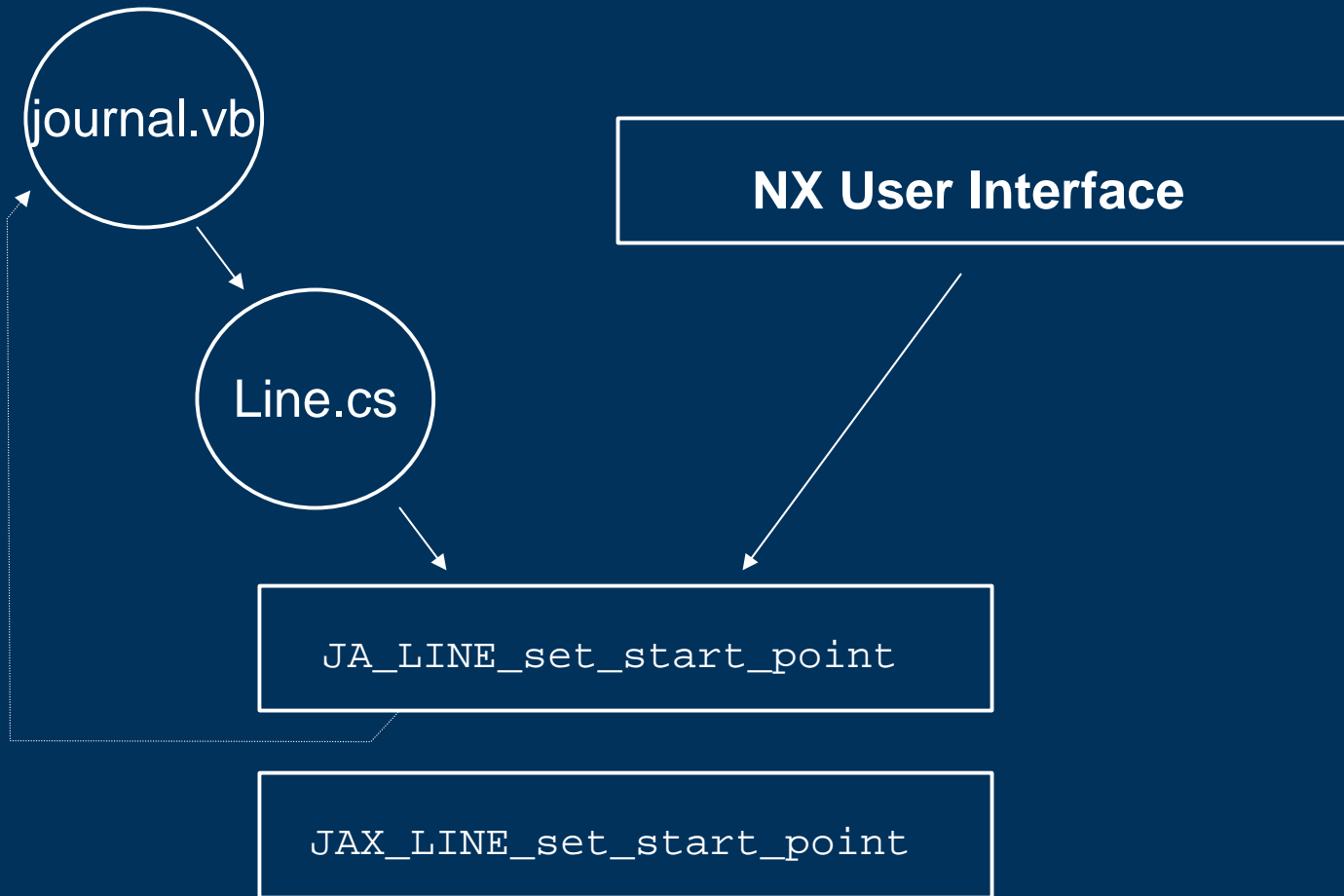
- ▶ Most User Function routines are callable from Java and .NET
- ▶ Provides access to functionality not exposed in the NXOpen API
- ▶ Allows you to start working in Java/.NET now
- ▶ Legacy applications can be ported
- ▶ UF wrappers generated by parsing UF header files
 - ▶ Some functions not suitable do to argument types.
E.g. void*



- ▶ Journal Recording:
 - ▶ Allows playback of an interactive session
 - ▶ Useful for generating template code for applications
 - ▶ Recording supported in all three languages
- ▶ Journaling layer provided by the compiler
 - ▶ Records all methods invoked, with input and output arguments
 - ▶ Separate journaling classes can write Java, VB or C++
- ▶ NX User Interface calls Journaling layer



NXOpen API with Journaling



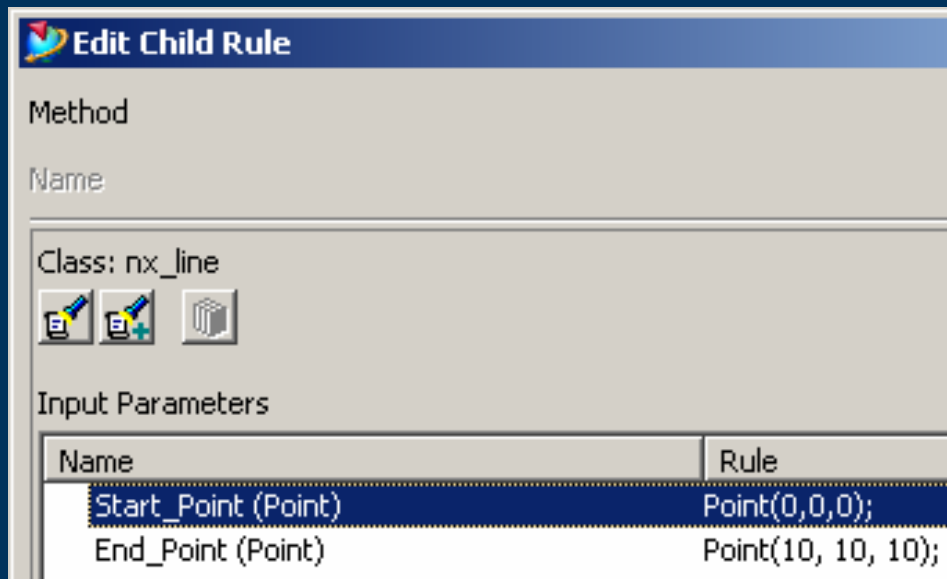


New in NX4

NX classes for Knowledge Fusion



- ▶ Knowledge Fusion allows developers to specify rules that capture design intent.
- ▶ KF classes are represented as a set of attributes
- ▶ New set of KF classes derived from JA files





Extrude Example






► In .NET

- ```
Dim extrudeBuilder As Features.ExtrudeBuilder
extrudeBuilder.Limits.StartExtend.SetValue("0")
extrudeBuilder.Limits.EndExtend.SetValue("25")
```

## ► In KF

Class: nx\_extrude

Input Parameters

| Name                                                                     | Rule                 |
|--------------------------------------------------------------------------|----------------------|
| <input checked="" type="checkbox"/> Limits_Limit_Opt (Name)              | nonsymmetric_offset; |
| <input checked="" type="checkbox"/> Limits_Start_Extend_Target (Any)     | nx_null();           |
| <input checked="" type="checkbox"/> Limits_Start_Extend_Value (Number)   | 0;                   |
| <input checked="" type="checkbox"/> Limits_Start_Extend_Trim_Type (Name) | Value;               |
| <input checked="" type="checkbox"/> Limits_End_Extend_Target (Any)       | nx_null();           |
| <input checked="" type="checkbox"/> Limits_End_Extend_Value (Number)     | 25;                  |
| <input checked="" type="checkbox"/> Limits_End_Extend_Trim_Type (Name)   | Value;               |
| <input checked="" type="checkbox"/> Limits_Symmetric_Option (Boolean)    | FALSE;               |



# Generating KF classes



- ▶ Original KF classes all written by hand
- ▶ New KF classes generated by the same compiler that produces .NET, Java and C++
- ▶ KF differs from procedural languages. Not all NXOpen classes are suitable.
- ▶ KF objects represented as a set of attributes. Not all NXOpen methods translate directly.
- ▶ Limited number of KF classes for NX4. More classes published in NX5.
- ▶ Still better than writing by hand



- ▶ NXOpen API is produced by compiling an interface definition file:
  - ▶ All API code is generated automatically.
  - ▶ UGS has no “preferred” language. Use the one that is best for *your* application.
- ▶ Journals are produced using the NXOpen API:
  - ▶ NX UI calls the same functions as external applications.
- ▶ Addition of KF support in NX4 demonstrates versatility of existing approach



[www.ugs.com](http://www.ugs.com)

Thank you